”Marchand” et ”non-marchand” dans l’économie des logiciels

Pierre-André Mangolte

« Mutations des industries de la culture, de l'information et de la communication »

colloque international des 25, 26 et 27 septembre 2006

"Marchand" et "non-marchand" dans l'économie des logiciels

Pierre-André MANGOLTE

CEPN (IIDE) - CNRS UMR n° 7115
Université PARIS-NORD
99, Av. Jean-Baptiste Clément
93430 VILLETANEUSE – FRANCE
e-mail : p.a.mangolte@wanadoo.fr
site web : http://perso.wanadoo.fr/lepouillou/

Août 2006

Introduction

Dans l'économie actuelle, marchande et capitaliste, cohabitent deux sphères, une sphère marchande et une sphère non-marchande. Une partie non négligeable de la production et des échanges de biens et services est en effet réalisée "hors-marché", souvent en dehors de toutes préoccupations commerciales. Il arrive même que les deux sphères co-habitent dans le même secteur, entremêlant alors production marchande et production non-marchande, vente et gratuité, recherche du profit et absence de motivation monétaire.

Il en est ainsi dans l'économie des logiciels. Un grand nombre de programmes (Apache, Linux, Firefox, etc...) sont produits quasiment sans versement de salaires, ni investissement, ni vente directe des produits; ceux-ci étant en règle générale distribués gratuitement. Ces logiciels libres (*free software*) ou logiciels *open source* reposent sur un ensemble de licences (GNU-GPL, licences BSD, MPL, etc.) qui sont de véritables innovations institutionnelles, réorganisant le système du *copyright* (ou droit d'auteur) appliqué aux logiciels en accordant des droits d'usage particulièrement étendus au détriement du droit patrimonial (*fructus*) du créateur du code. Ces licences ont permis l'émergence et l'installation dans la durée d'une certaine manière de produire, de transformer et de partager les logiciels. Une véritable économie des logiciels libres s'est ainsi progressivement mise sur pied en opposition aux stratégies "propriétaires" plus classiques des éditeurs de progiciels.

Parallèlement, un grand nombre de programmes sont depuis longtemps fournis gratuitement (comme *freewares*), sans pour autant être des logiciels libres ou des logiciels *open source*. On peut alors se poser la question de la "valeur des logiciels" en tant que produits - valeur marchande et/ou valeur d'usage - et aussi la question de leur "coût". Une question qui fait intervenir des déterminants techniques généraux (le logiciel comme code, "texte actif" ou fichier numérique), mais aussi la place du code considéré dans l'ensemble du système informatique, la définition particulière des droits de propriété intellectuelle, la relation existant entre le producteur et l'utilisateur de code (fourniture d'un produit ou relation de service, etc.), ainsi que les modèles économiques en présence.

Dans cette communication, nous voudrions analyser ces deux formes alternatives, marchandes et non-marchandes, de la production des mêmes objets techniques, les logiciels. Après avoir posé la question de la valeur des logiciels, et expliqué ainsi le phénomène "freeware", ou la mise dans le domaine public de certains codes, on cherchera à identifier les principales causes - techniques, sociales, institutionnelles - du clivage "marchand/non-marchand" dans la production et la distribution des programmes. Dans cette perspective, on s'intéressera plus particulièrement au

développement d'une économie des logiciels libres. On partira de l'innovation institutionnelle initiale (les licences) pour retracer la mise sur pied des grands projets *open source* à partir de réseaux étendus de programmeurs-utilisateurs. Dans une deuxième étape, un système de distribution (non-marchande ou marchande) est apparu, une distribution destinée aux utilisateurs finaux. Plus récemment, la simple cohabitation des logiciels libres et des logiciels sous droits de propriété classique (*copyright* et *patents*) cède la place à une articulation plus complexe des deux sphères, où l'on voit des logiciels libres remplacer des logiciels propriétaires, un abandon de certaines stratégies propriétaires antérieures, mais aussi l'intégration dans les modèles économiques des grandes firmes de l'industrie de l'informatique d'un certain nombre de programmes directement issus du mouvement *open source*.

I. Le logiciel, une valeur (marchande et d'usage) difficile à définir

L'économie des logiciels, de quoi parle-t-on ?

Dans une vision populaire et assez largement répandue, mais très peu scientifique, l'économie est naturellement monétaire et marchande. Autrement dit, toute activité de production de biens et/ou services accompagnée par des échanges semble devoir obligatoirement revêtir cette forme, et n'exister que s'il y a marché et activité commerciale, et même plus profondément activité financière. L'économie semble alors toujours impliquer le rassemblement d'un certain nombre de capitaux, petits ou grands, des opérations d'investissement accompagnées d'embauches, et une production vendue qui permettrait de dégager un profit; cet excédent étant la motivation principale, voire la condition et la cause même de l'existence de toutes les opérations de production et d'échange. Les seules exceptions à ce schéma sont pensées la plupart du temps comme résiduelles ou marginales, ou ajoutées après-coup à l'économie proprement dite. Ce sont les autoproductions domestiques (potager, bricolage, cuisine, etc.) ou encore les "gratuités socialement organisées[1]" des Etats modernes ("services publics" et opérations de redistribution).

Appliquer cette conception aux logiciels aboutirait à exclure de l'économie 70 à 90 % des activités de production de code et à ne retenir que ce qui est produit en vue d'une vente directe, séparée de toute autre fourniture, c'est-à-dire la production et l'édition des progiciels (*packages*) commerciaux ou des jeux vidéos, accompagnées par les stratégies propriétaires. Mais ce modèle économique n'a réellement pris son essor qu'à partir des années 1970; et a toujours été et demeure largement minoritaire dans l'ensemble des activités de production et de distribution de code. Le plus important reste en effet l'écriture lors des opérations de maintenance, de gestion des systèmes informatiques, d'administration de réseau, etc., lesquelles sont réalisées en interne (auto-production) ou par des sociétés de service, sans oublier la production à façon (ingénierie) ou l'écriture de code dédié, embarqué sur d'autres produits (Dréan, 1996; Campbell-Kelly, 2003; Horn, 2004).

Dans une version plus sophistiquée et un peu plus savante de l'économie, le découpage marchand / non-marchand est expliqué par les propriétés du bien ou service considéré qui fondent alors ce qu'on appelle un "échec de marché" (*market failure*). Des caractéristiques particulières du bien (cas de "l'information" par exemple (Arrow, 1962)) font que celui-ci est, en l'absence de protection juridique, facilement copiable, ce qui ruine toute mise sur le marché, toute vente, toute exploitation commerciale (problème du "*free-rider*"). L'existence d'un "échec de marché" explique alors économiquement, et justifie après coup, l'apparition ici d'un droit de propriété intellectuelle restrictif en matière d'usage. Pour certains économistes, partisans de ces droits de propriété intellectuelle (les *law and economics* de l'école de Chicago par exemple), tous les dispositifs juridiques regroupés sous ce terme (*patent* (brevet), *copyright* (droit d'auteur), marques, secrets de fabrication, etc.) obéissent à certains principes économiques communs, ayant pour finalité principale de permettre la création d'un marché, et d'accroître ainsi les incitations à la production en

1 Selon l'expression de Sagot-Duvauroux (2006).

facilitant les échanges et en permettant la division du travail (Besen et Raskind, 1991)[2].

Ces différentes approches ne peuvent cependant pas expliquer pourquoi certaines activités économiques tolèrent parfois dans la durée la coexistence d'une sphère marchande et d'une sphère non-marchande, et le fait par exemple que les mêmes biens et/ou services puissent être produits et échangés alternativement dans l'une ou l'autre sphère; ni expliquer les rapports que ces deux sphères entretiennent l'une avec l'autre. Or, c'est exactement le problème que pose l'économie des logiciels, non seulement actuellement, mais plus fondamentalement depuis l'origine de l'informatique.

On peut adresser deux critiques aux différentes approches évoquées ci-dessus :

(1) Le fait de ne prendre en considération que la valeur d'échange, une valeur qui n'apparaît que dans les opérations commerciales, sur le marché donc, et de ne définir l'économie qu'à partir de là. La valeur d'usage des biens considérés est largement laissée de côté, et négligée. Aucune analyse des fonctions que ces biens remplissent, des services qu'ils rendent, etc., n'est alors effectuée. Il en est de même pour les contraintes matérielles et techniques affectant la production et la distribution de ces biens. Dans les approches évoquées ci-dessus en effet, la valeur d'usage semble n'être là que pour permettre la création et la réalisation – une fois surmonté le problème de "l'échec de marché" – d'une valeur directement abstraite et monétaire (la valeur d'échange). Or partir des valeurs d'usage en économie est souvent fondamental. Rappelons que cette distinction entre valeur d'échange et valeur d'usage remonte à Adam Smith (1776), celui-ci définissant la "richesse des nations" comme "*toutes les choses nécessaires et commodes à la vie*", un ensemble de valeurs d'usages donc, et non comme "*les richesses inconsommables de la monnaie*", ce qui constitue l'erreur mercantiliste par excellence pour Smith (1776). L'économie – comme "science des richesses" – porte alors sur la création, la répartition et la consommation des valeurs d'usage.

(2) Le fait de négliger complètement l'analyse du cadre social et institutionnel de l'économie des biens et/ou services considérés, en postulant un cadre social inchangé quelle que soit l'activité, un cadre défini à partir d'une représentation standard de l'économie marchande capitaliste. Mais il peut y avoir ici, d'un secteur à l'autre, de grandes différences. Pour comprendre l'économie des logiciels par exemple, il faut manifestement analyser les relations nouées entre les producteurs (programmeurs) et les utilisateurs (programmeurs ou non) autour du logiciel, et même, plus généralement, les relations entre tous les protagonistes de l'économie des logiciels. Il faut s'intéresser aux règles et – problème qui est à l'origine du développement des logiciels libres – à la redéfinition du système des droits de propriété intellectuelle (*copyright* ou droit d'auteur) appliqué au logiciel.

Nous retiendrons donc – pour notre part – comme définition de l'économie celle d'Adam Smith, une définition dite substantive. Par économie des logiciels, on entendra tout ce qui concerne la production, la distribution (répartition, échange, etc.) et l'utilisation (concommation) des valeurs d'usage que sont les logiciels. Cette définition n'exclut pas l'analyse des activités non-marchandes, et l'analyse des rapports entre les deux sphères (marchande et non-marchande) de cette économie des logiciels. Elle inclut la production de logiciels comme produits (produits sur mesure ou progiciels, vendus ou livrés comme *freewares*) et les services logiciels, si importants dans cette économie, tout comme l'auto-production chez les utilisateurs[3].

---

2 Dans les versions les plus extrêmes (populaires ou savantes) de ce raisonnement théorique, l'absence de marché, tué par la "piraterie" signifie même la fin de toute production (ou création). Pour d'autres, c'est simplement un problème d'incitation et d'optimisation. Mais pour toutes ces approches, c'est ce problème économique qui est censé justifier économiquement l'existence d'un droit de la propriété intellectuelle, à l'exclusion de toute autre explication directement juridique ou historique par exemple.

3 J'éviterai donc le terme "industrie" qui évoque trop la production des "biens", et laisse de côté les services. Ainsi Dréan (1996) se refuse à parler d'une industrie des logiciels et réserve ce terme aux progiciels (produits logiciels). Campbell-Kelly (2003) l'utilise essentiellement pour la sphère marchande de l'économie des logiciels. Classer l'activité de production des logiciels dans "l'industrie" ou les "services" a d'ailleurs toujours posé problème aux statisticiens... et aux professionnels de l'activité eux-mêmes.

Nous allons maintenant reprendre l'analyse de la valeur du logiciel sous l'angle du coût de production (dépenses à engager et temps de travail), puis nous intéresser aux deux valeurs d'usage du logiciel, et aux deux demandes sociales qu'on peut distinguer ici. Nous laisserons de côté, provisoirement, la redéfinition du *copyright* et l'apparition du *copyleft*, des licences open source, et le mouvement des logiciels libres, dont nous parlerons dans la deuxième partie.

Contraintes et coûts de production

Le logiciel est un ensemble d'instructions destinées à une machine (ordinateur) qui prend la forme d'un texte écrit dans un langage de programmation (le code-source). Ce texte est ensuite transformé (au moyen d'un compilateur en général) en code-objet (les instructions en langage machine). Le logiciel est donc un objet technique, une sorte d'outil fait de signes, un écrit destiné à fonctionner sur (et avec) une machine physique (le *hardware*), et souvent avec d'autres logiciels. Il doit alors respecter toutes sortes d'exigences techniques qui contraignent son écriture et son développement.

Produire un programme est tout d'abord un travail purement intellectuel de conception et d'écriture. Il faut définir le projet et son architecture, et mettre en forme le code source. Il faut ensuite tester le programme et inévitablement le corriger, en ré-écrivant certaines parties du code, puis le re-tester, etc. Le code ne vaut en effet que par son caractère fonctionnel. Mais il n'existe malheureusement pas de "preuve" en matière de code, et même une relecture minutieuse de celui-ci ne suffit pas à garantir qu'un logiciel soit exempt de bogues ou d'erreurs, et conforme à ce que l'on souhaite obtenir. Le programme doit donc être essayé sur une machine, dans toutes les configurations possibles. La seule "preuve" possible étant de nature empirique, il faut toujours essayer (et ré-essayer) le code. Cette phase des tests peut conduire à utiliser ou écrire d'autres programmes spécifiques (code non fonctionnel), une activité qui représente 50 à 70 % du coût réel d'un projet informatique (Printz, 1998). Il est préfèrable alors de parler du "développement" d'un logiciel, plutôt que de sa "production". Le développement du logiciel se prolonge ensuite dans des essais *in situ* faisant intervenir les différentes manières dont les utilisateurs finaux interagissent avec les commandes du système informatique, puis dans des opérations de maintenance, qui là encore conduisent à retravailler le code source : correction de bogues, ajouts d'extensions, etc.

Ces activités sont souvent longues et coûteuses, coûteuses en termes de dépenses et surtout de temps de travail (salarié ou non), mais ce coût est supporté une fois, et une seule fois, pour la production d'un logiciel[4]. On ne parle ici évidemment que la production au sens strict, à l'exclusion de tout coût marketing et de distribution, dans l'éventualité où le programme serait commercialisé. En effet, dans ce cas, le coût de production et de développement proprement dit ne pèse en général que 15 à 20 % du total, les frais de marketing représentant de 35 à 50 %, et le reste étant le coût de la distribution physique des paquets logiciels (Campbell-Kelly, 2003). Mais pour la simple production de la valeur d'usage, le plus important est le temps de travail des programmeurs (et la durée); les outils utilisés au cours du développement étant – en dehors de l'ordinateur – essentiellement les langages et des outils logiciels : éditeurs de code, compilateurs, etc[5].

Le logiciel est donc une oeuvre unique, dont on peut cependant réaliser autant de copies à l'identique qu'on le souhaite par simple duplication (copie du code source initial en version papier, bandes magnétiques, disquettes, téléchargement), cette duplication ne supprimant pas et n'affectant pas l'oeuvre d'origine. On a bien ici deux processus de production distincts et séparés: (1) la

4 Un système d'exploitation comme Linux représente environ 3 millions de lignes de code (2002), plus de 2200 programmeurs ayant participé à son développement (Gosh et David, 2003). Pour comparaison, Windows 95 "pesait" environ 1,5 millions de lignes de code. Une évaluation du coût par analogie avec une structure propriétaire classique (modèle COCOMO) donne alors 945 hommes-années, et un coût monétaire de 127 millions de $. Mais bien évidemmment, ces sommes n'ont jamais été réunies ni dépensées ici.

5 Ces outils logiciels nécessaires au travail de programmation peuvent être produits par ceux qui les utilisent, par les programmeurs eux-mêmes, et ont été d'ailleurs historiquement parmi les premières réalisations du programme GNU de logiciels libres (Emacs, Gcc, etc.).

production et le développement initial du logiciel; (2) la duplication à un moment donné de ce programme, celui-ci étant considéré comme achevé, suffisamment stable du moins. Cette situation est complètement différente de celle qui existe dans l'industrie des "biens", industrie automobile ou aéronautique par exemple. Dans ces industries, les coûts de conception et de développement d'un modèle, jusqu'à la mise en production, peuvent être considérables. Mais, une fois cette phase achevée, il faut encore produire chaque véhicule séparément, un à un; et chacune de ces productions est une opération compliquée et coûteuse, qui n'a rien à voir avec la simple duplication d'un fichier numérique.

En matière de logiciel, la duplication est généralement peu coûteuse, limitée la plupart du temps au coût d'un support (bande magnétique, disquettes, CD, etc.) et au temps d'enregistrement, auquel il faut ajouter cependant la documentation. Cette duplication est de moins en moins coûteuse aujourd'hui avec l'apparition des téléchargements utilisant l'internet, le coût étant alors directement supporté par celui qui réalise la copie. Comme la duplication ne dépossède pas le détenteur initial du programme ("non-rivalité" ici) et que cette duplication ne lui coûte rien en cas de téléchargement, la possibilité économique d'une diffusion à très grande échelle et non-marchande des logiciels s'ouvre alors. C'est ce qui explique l'apparition et l'existence permanente de *freewares* (ou graticiels), ces produits logiciels qui, quelque soit le type de licence qui les accompagne (voir plus loin), sont fournis gratuitement.

Les graticiels n'ont rien à voir en principe avec les logiciels libres. On peut trouver en effet sous forme de graticiel le code-source d'un logiciel libre ou (de plus en plus) une version binaire du même logiciel, mais tout aussi facilement la version binaire d'un programme sous licence propriétaire classique sans fourniture du code-source. Leur existence témoigne simplement du fait que le financement (et le développement) du programme est déjà réalisé quand la duplication arrive, et que cette opération - à coûts quasi-nuls pour le détenteur du fichier - est totalement séparée du développement proprement dit, ce qui est un fait technique et non institutionnel.

Une économie stable de production de graticiels est possible car le coût de développement d'un logiciel (temps, travail et dépenses éventuelles) ne doit être supporté qu'une fois, et peut être financé à fonds perdus, ou indirectement. Et même si on laisse de côté les nombreux programmes financés par des institutions non-marchandes (universités, administration, fondations, etc.) ou réalisés par des programmeurs sur leur temps libre (un temps de travail non payé), il y a place pour ce type de production dans la sphère marchande de l'économie. Le financement indirect du développement des logiciels est d'ailleurs une vieille tradition. Jusqu'aux années 1970 en effet, rares sont les programmes vendus en tant que tels. Deux idées communes résument alors assez bien la situation : (1) le logiciel est coûteux à produire, mais n'a pas réellement de prix (ou de valeur marchande); et (2) le logiciel ne peut avoir de valeur propre car il relève d'une activité de service. Les situations de production du code et des programmes sont encore si variées que ces deux idées sont souvent encore vraies.

Même aujourd'hui, où existe pourtant une véritable industrie des progiciels, le code écrit en vue d'une vente directe ne représente qu'une toute petite partie de l'univers de la progammation (Raymond, 1999). Le code écrit en interne dans les banques, les compagnies d'assurances ou les entreprises, n'a ainsi souvent pas réellement de valeur marchande et n'est d'ailleurs pas vendu. L'activité de programmation est alors financée indirectement par la commercialisation des produits ou services spécifiques de ces firmes. D'autres logiciels sont développés pour accompagner la vente de machines (imprimantes, appareil photos, etc.), et sont invendables séparément (les pilotes par exemple). C'est la vente de ces machines (et des consommables) qui finance alors le développement du logiciel. D'autres enfin sont produits ou livrés dans une relation de service et peuvent alors faire partie d'une offre globale.

Les deux valeurs d'usage du logiciel

En matière de valeur d'usage, on a pu constater qu'il y avait deux usages possibles du logiciel,

avec ici deux sortes d'utilisateurs. Il y a d'abord l'utilisateur final essentiellement intéressé par les fonctionnalités d'un programme et par une version tournant effectivement sur son propre système informatique. Si le programme tourne bien, ou suffisamment bien, cet utilisateur final n'est guère intéressé par l'accès au code-source, et ceci d'autant plus qu'il est en général incompétent en matière de programmation. Une version binaire compilée pour son système d'exploitation lui suffit, et c'est pour satisfaire cette demande sociale des utilisateurs finaux qu'ont été conçus les progiciels (et les packages de produits-logiciels). La production et l'utilisation des programmes sont alors complètement séparées, un schéma qui existe aussi pour la plupart des autres biens (automobile, pain, etc.) mais qui en matière de logiciel apparaît souvent inadapté aux situations de production et d'utilisation.

Il existe en effet une autre demande sociale et une autre catégorie d'utilisateur, l'utilisateur-programmeur. Celui-ci n'est pas simplement intéressé par par l'exécution d'un programme, mais veut pouvoir le modifier le cas échéant en ré-écrivant une partie du code-source. Il s'agit par exemple d'une simple amélioration (correction d'un bogue), ou d'ajouter une extension, ou de transférer le programme sur un autre système informatique, etc. Il arrive même que le logiciel soit considéré comme un point de départ utile pour produire un autre logiciel, dans une perspective d'innovation et de changement techique. Le code-source est en effet composé d'un ensemble de modules, de pièces et d'éléments qui peuvent parfois être ré-utilisés pour développer, en économisant ainsi du travail de programmation, un programme complètement différent du premier. Ici, l'accès au code-source et la possibilité de modifier (légalement) le code deviennent primordiaux, et un tel utilisateur-programmeur ne peut se satisfaire d'une version binaire. On voit donc apparaître ici la deuxième valeur d'usage du logiciel. La production et l'usage restent alors étroitement mêlées, et le logiciel (comme code-source) est avant tout considéré comme l'objet éventuel d'un travail de programmation.

Trois points importants méritent alors d'être soulignés :

(1) Un logiciel est toujours un produit imparfait et la correction des bogues une nécessité pour conserver au programme sa valeur d'usage en terme d'exécutable. Quand le programme est livré comme progiciel sans accès possible au code-source, l'utilisateur-programmeur qui pourrait parfois facilement corriger le programme, ne peut le faire. Il doit attendre que le producteur-éditeur (sans lien direct avec lui) découvre le bogue, le corrige, et livre enfin une nouvelle version ou une mise à jour (*update*) pour voir le défaut disparaître.

(2) Un logiciel n'est jamais réellement "consommé" ou usé; il ne s'abîme pas, sauf accident, quend on l'utilise. Mais sa durée de vie est limitée car l'environnement change : les systèmes informatiques se modifient, la demande des utilisateurs finaux inévitablement se transforme. Le logiciel, sous peine d'être assez vite frappé d'obsolescence, doit donc être redéfini et réécrit, ce qui implique là-encore possibilité d'accès au code-source.

(3) Le logiciel est destiné à un système informatique. Il ne peut réellement fonctionner que dans un système plus vaste comprenant une machine physique et d'autres programmes (système d'exploitation, bibliothèques, navigateurs, etc.), plus l'intervention éventuelle d'utilisateurs humains. Cela soulève le problème des interfaces, des formats, des protocoles, des normes, des standards, qui assurent l'interconnexion technique et le fonctionnement systémique de l'ensemble. En matière de logiciels, cela signifie qu'il faut une définition commune de tous les éléments qui assurent l'interconnexion. Il faut aussi que cette définition reste inchangée le plus longtemps possible pour permettre le maintien au cours du temps des interconnexions. Il faut enfin qu'elle soit connue par tous ceux qui en ont besoin pour produire les différents éléments du système. Une partie du code-source, là encore, doit être dévoilée et rendue disponible aux autres.

Ce problème technique d'interconnexion débouche alors sur un problème de coordination sociale. Il faut un accord sur la définition de l'interface et sa stabilité (fixation d'une norme par exemple); si cet accord est trouvé, la coordination est possible et chaque programmeur peut travailler isolément, indépendamment des autres, en respectant simplement la définition de

l'interface. Le logiciel, ou le module (Narduzzo et Rossi, 2003), ainsi développé sera compatble avec les autres, et l'ensemble fonctionnera parfaitement bien. Sinon, rien ne marchera. On peut alors trouver ici des formats, des protocoles, des normes d'interconnexion propriétaires ou non, "ouvertes" (*open*) ou "fermées". Un format est dit propriétaire s'il a été mis au point par une firme, qui l'a introduit et contrôle son évolution ultérieure. Le propriétaire peut alors le garder fermé – les spécifications étant cachées, couvertes par le secret de fabrication, et l'utilisation par d'autres rendues impossible ou difficile (dans certains cas en effet, la législation (européenne par exemple) autorise ici la décompilation). Mais il peut aussi le mettre dans le domaine public, en donnant toutes les spécifications de l'interface, et en autorisant largement son utilisation, y compris par des concurrents[6]. On a alors un format propriétaire "ouvert". Les éléments nécessaires aux interconnexions techniques peuvent aussi appartenir directement au domaine public, et sont alors automatiquement "ouverts" (comme la norme POSIX pour les systèmes d'exploitation UNIX, les protocoles et les langages de l'internet, les productions du w3c, etc.).

Il existe ici évidemment des stratégies propriétaires pour garder secrètes certaines interfaces afin d'évincer les adversaires, d'empêcher la production de programmes alternatifs, ou d'enfermer les clients dans une relation de dépendance. Mais la dépendance systémique étant souvent symétrique, ces stratégies ne sont pas toujours des plus pertinentes. Et par ailleurs, il existe aussi une pression sociale importante et permanente dans l'industrie informatique pour que soient maintenus "ouverts", et mis de fait en propriété commune, tous les éléments qui font interface, une préoccupation qu'on retrouve dans les logiciels libres, mais qui déborde largement le clivage entre ces logiciels et les stratégies propriétaires, un clivage dont dont nous allons parler maintenant.

II. Les logiciels libres dans l'économie des logiciels

Logiciels libres et stratégies propriétaires

A partir du milieu des années 1960, sont apparus aux Etats-Unis les premiers produits logiciels (packages), un marché qui prendra progressivement son essor dans les années 1970-1980[7]. A la différence de tout ce qui existait jusqu'alors, le logiciel est alors vendu comme un produit séparé, indépendamment de la vente d'une machine ou d'un service. Un débat sur la protection juridique la plus adaptée à cette nouvelle forme de commercialisation s'engage alors dans la profession. Les *patents*, conçus dans beaucoup de législations comme devant protéger des artefacts tangibles, apparaissent largement inadaptés; et le *copyright*, admis par le Congrès dès 1964 avec des conditions restrictives, protége assez mal le producteur du logiciel, à partir du moment où la simple lecture du code dévoile les algorithmes et les fonctionnalités (permettant donc une ré-écriture différente du programme). La stratégie retenue combine alors le secret de fabrication, c'est-à-dire la commercialisation de binaires (à l'exclusion du code-source) accompagnée de clauses de non-divulgation pour le personne et d'un système de licences restrictives en matière d'usage (nombre de machines, etc.) et de redistribution, plus le *copyright* pour la documentation. Le but ici est évidemment la maximisation des recettes (et de la rente), en protégeant le plus longtemps possible cet actif commercial qu'est devenu le logiciel. C'est aussi la naissance historique de la valeur d'échange pour les logiciels.

Le code-source est alors caché et les licences interdisent toute opération de *reverse engineering* et/ou de décompilation, ainsi que la modification du code fourni exclusivement en

6 Conserver "fermé" un format (ou une interface) n'est pas toujours une bonne politique, car sa valeur commerciale dépend souvent de manière cruciale de son adoption. L'histoire du format pdf est significative, son utilisation n'ayant réellement décollée qu'à partir du moment où il a été ouvert et mis dans le domaine public, et de plus Acrobat Reader distribué en graticiel. Cette décision a créé alors un marché pour les autres produits d'Adobe utilisant le pdf (Acrobat Distiller, etc.), et permis à des sociétés concurrentes (et à l'*open source*) d'utiliser le format.

7 Cet essor est favorisé par la décision d'IBM en 1968, sous pression des autorités anti-trust, de dégrouper (*unbundling*) l'ensemble de son offre. IBM abandonne alors les ventes liées en séparant désormais dans ces facturations l'ingénierie de système, la formation, la maintenance, les services de programmation et la fourniture de logiciels (Campbell-Kelly, 2003).

binaire. Le code est fermé et des "murs de plus en plus hauts séparent désormais les différents programmeurs", selon l'expression de Richard Stallman (1998). Les pratiques antérieures de circulation et de partage du code sont proscrites, et la coopération entre programmeurs n'appartenant pas à la même firme interdite. Ces nouvelles règles entrent en contradiction directe avec les normes de comportement largement répandues à l'époque dans la profession, avec des habitudes de re-distribution libre des programmes (y compris pour des versions en cours de développement), de partage des modifications, d'entraide et de coopération. Un travail collectif et coopératif s'était ainsi plus particulièrement mis en place entre les Bell Labs et différentes universités autour du premier système d'exploitation Unix (Salus, 1994; Di Bona, 1999). C'est pour contrer une évolution qui menaçait ce processus d'invention collective (Allen, 1983) et plus généralement "toute communauté coopérative", que furent inventées les premières licences *open source* (BSD et GPL), puis les logiciels libres, avec la Free Software Foundation (1985) et le projet GNU (*GNU's Not Unix*) (Stallman, 1998).

Dans les licences *open source*, le propriétaire des lignes de code (détenteur du *copyright*) accorde un droit d'usage particulièrement étendu à l'utilisateur. Celui-ci peut copier librement le programme, le modifier comme il l'entend, et le redistribuer à sa guise. Il obtient pour cela un libre accès au code source. On a évidemment ici un dispositif conçu avant tout pour les programmeurs-utilisateurs. Les droits accordés concédés dépassent en effet largement la demande d'un utilisateur final, le simple droit d'utilisation d'un programme sans le modifier. Ils reposent sur l'idée que les programmes ont vocation à être utilisés, ré-utilisés et développés par leurs utilisateurs, lesquels doivent pouvoir pour cela accéder librement au code-source.

Dans la licence GNU-GPL (*General Public Licence*)[8], la redistribution du code est soumise à une condition particulière qui interdit sa transformation en logiciel propriétaire. La licence impose en effet une redistribution dans les mêmes conditions que la licence d'origine, avec fourniture du code-source, même modifié, accompagné par les mêmes droits d'usage que le code d'origine. Un logiciel libre ne peut alors être incorporé au sein d'un logiciel propriétaire (mais peut très bien être distribué avec un logiciel propriétaire dans un ensemble (*packages*) de logiciels, accompagnés chacun de leur licence respective). La GPL utilise la loi du *copyright* en la détournant de son utilisation habituelle (c'est le *copyleft*). On a là une véritable innovation juridique et institutionnelle qui sécurise à l'avance le développement des logiciels libres et permet d'installer dans la durée une certaine façon de produire et de développer le code. A partir de là est né le projet GNU dont l'objectif est d'encourager la production progressive d'un ensemble de programmes sous licence GPL, portables sur n'importe quelle machine et comprenant entre autres un système d'exploitation Unix. Il s'agissait, selon la formule de Mélanie Clément-Fontaine (1999) de "créer progressivement un fonds commun de logiciels libres dans lequel tout le monde pourrait puiser, auquel chacun pourrait ajouter, mais duquel personne ne pourrait retrancher".

Ce fonds commun, protégé de toute privatisation ultérieure par la clause *copyleft* peut alors se constituer et grandir progressivement, de manière cumulative et sans ordre particulier, en fonction des apports des uns et des autres. "Au commencement, raconte Stallman (1998), il s'agissait d'écrire n'importe quel composant, car tous les composants manquaient. L'ordre importait peu alors. Plus tard, une liste de ce qui manquait fut établie, la *task-list* de GNU; cette liste permettait de recruter des développeurs pour telle ou telle partie du projet". Vers 1990, tout le système existait. Seul manquait encore le coeur du système d'exploitation. La conception technique retenue par la FSF (le projet Hurd basé sur un principe de micro-noyau) prenant du retard et s'avérant en fait difficile à réaliser, c'est finalement un noyau compact plus classique écrit par Linus Thorwald en 1991 (Linux), qui mis sous GPL et largement développé depuis, a permis de réaliser ce qui était le but même du projet GNU, un système complet de logiciels libres.

8 Pour une définition précise des différentes catégories de licences (logiciel libre, logiciel copylefté ou non-copylefté, logiciel *open source*, etc.) voir le site de la Free Software Foundation (http://www.fsf.org/licensing); mais aussi Perens (1998). Parmi toutes les licences *open source*, la GPL est la plus utilisée : 86 % des logiciels pris en compte dans la base WIDI (Robles et alii., 2001).

Le "gratuit" et les logiciels libres

Contrairement à une idée qu'on peut rencontrer çà et là, un logiciel libre n'est pas automatiquement et obligatoirement un *freeware* (ou graticiel)[9]. Il n'existe en effet aucune clause dans les licences *open source* – et en particulier dans la licence GPL – qui impose de distribuer le code-source gratuitement, même quand on l'a reçu comme graticiel. La définition du logiciel libre (*free software*) donné par la Free Software Foundation ou du logiciel *open source* donné par l'Open Source Initiative (Perens, 1988[10]) n'incluent pas cette obligation, qui limiterait d'ailleurs inutilement la liberté d'usage accordée par ce type de licence aux utilisateurs. Il n'est donc aucunement interdit de vendre (ou de revendre) le code-source. Il est interdit par contre d'interdire aux autres de choisir dans ce domaine entre la vente ou la gratuité; et une licence qui imposerait une revente ou une livraison gratuite du logiciel (code-source) ne pourrait considérée comme une licence de logiciels libres. La licence met cependant tous les distributeurs du code en concurrence, et cette règle empêche toute émergence à moyen ou long terme de situation de prélèvement de rente. Le prix réellement exigible ne peut donc dépasser, sauf cas particuliers[11], le coût de production de la distribution (bandes magnétiques, disquettes, commercialisation d'un *package*, etc.) ou le paiement des services qui peuvent accompagner un programme fourni de fait gratuitement (aide à l'installation, formation, maintenance, etc.). La forme la plus répandue et la plus rationnelle de distribution est alors, quand celle-ci se fait en utilisant l'internet, le téléchargement en graticiel, une conséquence indirecte des licences, qui n'est cependant pas exigée par celles-ci.

De toute manière, en adoptant une licence *open source*, les différents programmeurs qui utilisent et transforment le code ont renoncé aux droits patrimoniaux qui accompagnent généralement le *copyright*, et abandonné toute idée de rivalité commerciale construite sur le caractère intangible de leur code. Chaque auteur, propriétaire d'un certain nombre de lignes de code, renonce en effet à contrôler ce que deviendra sa propre production, tant qu'elle reste du moins en propriété commune (clause *copyleft*), en la rendant ainsi disponible pour l'invention collective. Car la finalité même d'une licence *open source* est bien de permettre l'évolution des programmes et le travail sur le code, de favoriser l'innovation, en écartant ici toute tentation et stratégie propriétaire. La licence élimine donc formellement tout conflit sur les usages du code et place tous les utilisateurs (et programmeurs) sur un pied d'égalité. Des formes de coopération peuvent alors apparaître, se pérenniser et déboucher sur des productions communes. Dans une organisation plus traditionnelle (un éditeur de programme attaché à une stratégie propriétaire par exemple), c'est le rapport salarial qui, faisant de la firme le seul propriétaire du code, neutralise les effets destructeurs pour tout travail en coopération du système des droits de propriété individuels. Ici, c'est la micro-institution des licences *open source* qui règle le problème.

Le but est la production d'une valeur d'usage, avec toutes les ambiguités de la valeur d'usage en matière de logiciels, valeur d'usage pour l'utilisateur final et/ou valeur d'usage pour les programmeurs eux-mêmes, pour des versions en cours de développement. La motivation n'est donc pas l'enrichissement individuel ou la course au dollar, mais une autre forme d'intérêt comme le montre toutes les enquêtes sur la question, un intérêt directement lié au métier ou à la situation d'utilisateur-programmeur. C'est par exemple un intérêt direct pour l'amélioration d'un logiciel qu'on utilise soi-même. C'est parfois aussi le défi purement individuel du problème technique à résoudre,

9 On peut ainsi lire dans le livre de F. Lévêque et Y. Menière, *Economie de la propriété intellectuelle*, collection Repères (2003), la phrase suivante : "Comme leur nom l'indique, les logiciels libres sont disponibles gratuitement (sic)" (p. 47). Et pourtant, il suffit d'aller sur le site de la FSF pour lire : "L'expression «Logiciel libre» fait référence à la liberté et non pas au prix. Pour comprendre le concept, on doit penser à «liberté d'expression», et non à «entrée libre»."

10 Cf aussi "http://www.fsf.org/licensing" et "http://www.opensource.org/docs/definition.php".

11 Un paiement direct à l'auteur du code par exemple, ou à l'équipe qui a produit le programme, dans une logique de subvention ou de contribution (financière) à un projet en développement et non dans une logique d'achat d'un produit réellement commercial. On peut alors avoir deux distributions qui cohabitent, une "marchande" et une "gratuite".

ou plus simplement un moyen d'apprentissage, à partir du moment où le travail sur le code et la pratique des tests sont les seuls moyens qui permettent réellement d'apprendre à programmer. Tout cela étant renforcé par la conviction que la production dans le cadre de l'*open source* permettra effectivement d'obtenir des logiciels qui tournent bien, avec des fonctionnalités qui s'amélioreront peu à peu (Yamagata, 1997), ou par l'adhésion aux valeurs du partage et du *copyleft* et à la philosophie du logiciel libre (Stallman, 1999).

Dans les licences *open source*, le logiciel est essentiellement un paquet de code-source qui doit pouvoir circuler, passer de l'un à l'autre, et se transformer ainsi progressivement. Sa vente éventuelle est un problème secondaire, en pratique restée marginale. Les licences ne lient pas recettes et financement, et empêchent plutôt d'établir une liaison étroite entre les deux. Les modes de financement du développement des logiciels libres ne peuvent alors – comme pour les autres graticiels – qu'être à fonds perdus ou indirects. Des fondations (à but non lucratif) sont par exemple constituées pour réunir des fonds aux origines diverses : dons, contributions de certaines entreprises de l'industrie informatique, etc. Ces fonds financent le travail salarié et les dépenses qui sont nécessaires au développement de certains projets. Dans d'autres cas, ce sont simplement des dons en code-source et le travail gratuit d'un grand nombre de programmeurs volontaires qui assure le "financement". Un grand nombre de bénévoles en effet contribuent à faire vivre le mouvement des logiciels libres, comme programmeurs ou autrement[12].

Invention collective et coopération sociale

On peut produire du logiciel libre dans le cadre d'une organisation traditionnelle, avec une équipe salariée stable, une définition préalable de ce qui doit être produit, un plan, un ordonnancement précis des tâches et, le moment venu, la livraison d'un produit fini, ou supposé fini. C'est pourtant une situation relativement exceptionnelle. On peut aussi produire un logiciel libre de manière solitaire (ou presque). Un programmeur isolé rédige son code-source, le compile, le teste et le met sous licence GPL à la disposition de tous, en intégrant ensuite les éventuelles corrections de bogues qu'on lui envoie ou en travaillant sur ceux qu'on lui signale. Il n'y a rien en effet dans les licences *open source* qui interdit de procéder ainsi, rien qui définit la manière dont on doit fabriquer le code, rien qui impose la coopération avec d'autres dans la production et le développement d'un programme. Et d'ailleurs, cette dernière situation – la production solitaire - est de fait très fréquente. C'est même presque la règle pour les programmes dont la taille – en termes de lignes de code – n'est pas très importante. Ainsi dans une étude parue en 2002, Krishnamurty a pu montrer que plus de 50 % des projets actifs et matures recensés par la base de données "sourceforge", étaient le fait d'une seule personne; c'est le modèle de la "cave". D'autres par contre impliquaient des équipes plus larges (29 % avaient mobilisés plus de 5 développeurs, avec un pourcentage du même ordre pour les discussions autour du programme) (Krishnamurty, 2002).

Et pourtant, la possibilité ouverte par ce type de licence de partager, de modifier et de redistribuer librement, a permis de retrouver les principes de coopération sociale qui existaient déjà pour le développement des premiers systèmes d'exploitation Unix (Salus, 1994; Di Bona et alii., 1999). On a ainsi vu apparaître des formes de production originales, faisant intervenir de nombreux programmeurs dispersés un peu partout dans le monde et coopérant volontairement pour développer tel ou tel projet logiciel[13], comme par exemple les grands projets *open source* : Apache, Linux, Gnome, Mozilla, etc. A la différence d'une organisation traditionnelle, il n'y a ici ni délais fixés à l'avance, ni répartition réellement pré-établie des tâches, ni coordination marchande, ni hiérarchie,

12 Environ 20 % des programmeurs contribuant au développement des logiciels libres sont payés pour le faire, les autres sont ici bénévoles, consacrant pour la plupart en moyenne quelques heures par semaine à cette activité. Cf. enquête WIDI (Robles et alii., 2001) et Kim (2003).

13 En analysant le code et les crédits des différentes versions de Linux de 1994 à 2002, on a pu identifier 158 auteurs (1994), 618 (1997), et 2263 (2002), soit une communauté largement dispersée géographiquement dans toute l'Europe, en Amérique du Nord et ailleurs dans le monde (la version la plus récente intégre évidemment du code (et des auteurs) présent dans les versions antérieures).

ni autorité découlant d'une relation salariale. A l'inverse cependant du modèle de la "cave", ces projets mobilisent un grand nombre de programmeurs. Le travail peut alors être largement distribué géographiquement, les communications entre développeurs et les transports du code étant réalisés aux moindres frais grâce à l'internet.

A l'origine de ce type de projet, il y a toujours un certain paquet de code, un logiciel donc à modifier ou à faire évoluer. Il y a aussi un fondateur, une personne ou une équipe qui prend en charge la collecte et l'intégration des contributions dans l'ensemble du code-source en voie de transformation. On parle alors du (ou des) "mainteneur(s)". Ces mainteneurs écrivent souvent eux-mêmes une partie du code, mais leur rôle principal est plutôt de filtrer les contributions afin de conserver et d'améliorer progressivement la qualité du programme. Le projet repose alors sur un principe de tri à l'entrée des contributions avant leur inscription dans le code-source retenu, avec une sortie totalement ouverte, n'importe qui pouvant venir télécharger librement le code des différentes versions en développement. De tels projets ont bien un début, mais n'ont aucune fin programmée. Ils évoluent de manière cumulative et incrémentale, en fonction des problèmes rencontrés et de l'arrivée des différentes contributions; et le rythme du développement peut s'accélérer brutalement quand le nombre d'utilisateurs-programmeurs intéressés et mobilisés augmente.

L'essor de ces formes de production de logiciels libres s'explique aussi par l'importance de la relation utilisateur-programmeur dans l'économie des logiciels. Les "communautés" du logiciel libre n'étant que des exemples particuliers de production collective de l'innovation par des "réseaux d'utilisateurs-producteurs" (Von Hippel, 2002). Pour qu'un tel réseau d'utilisateurs, plus précisément d'utilisateurs-producteurs, apparaisse en matière d'innovation, il faut trois conditions : (1) Certains utilisateurs au moins sont incités à innover; (2) certains utilisateurs au moins sont incités à révèler leurs innovations; (3) la diffusion des innovations par les utilisateurs est peu coûteuse et peut concurrencer le mode commercial de production et diffusion des innovations. Quand ces trois conditions sont réunies, l'évolution de la technique (du logiciel par exemple) est pilotée directement par les utilisateurs qui produisent eux-mêmes les transformations nécessaires. Le produit et la valeur d'usage évoluent alors sans aucune médiation marchande et sans le filtre des services marketing des firmes productrices de la technologie[14]. Or, ces trois conditions sont facilement réunies en matière de production de logiciel à partir du moment où les problèmes posées à la coopération par les droits de propriété intellectuelle ont été règlés d'une manière ou d'une autre, en particulier par l'adoption d'une licence de logiciel libre. Le réseau d'utilisateurs peut d'ailleurs exister en dehors de tout système de licence. Ainsi, nombreux sont les forums d'utilisateurs dans l'univers des logiciels propriétaires, et dans ces forums, on produit de la formation, de l'aide en ligne, etc. Mais pour que cette coopération sociale, plus ou moins spontanée, se transforme en production de logiciels, il faut nécessairement pouvoir accéder au code-source, avoir ici des utilisateurs qui soient aussi des programmeurs, et disposer de règles de comportement claires vis-à-vis du code, ce qui est permis par les licences. Quand un réseau d'utilisateurs prend conscience de sa spécificité, de son existence en tant que groupe, et développe un fort sentiment d'appartenance, on peut même parler d'une "communauté", un élément qui renforce d'ailleurs fortement la motivation (Von Hippel, 2002[15]).

L'économie des logiciels libres et l'industrie informatique

Dans l'économie des logiciels libres, les activités de développement et de travail sur le code sont maintenues à l'écart – en règle générale – des problèmes de financement et des préoccupations

14 Dans certaines industries, la liaison utilisateur-producteur est fondamentale pour le changement technique, les principales innovations étant introduites directement par les utilisateurs ou en liaison avec eux; voir l'exemple des machines-outils (Rosenberg, 1976). Von Hippel donne des exemples d'innovation majeures introduites par les utilisateurs eux-mêmes, les planches de surf, l'instrumentation scientifique, des machines nécessaires à la production des semi-conducteurs, etc.

15 Un bon exemple de projet de développement logiciel pris en charge directement par des utilisateurs est Apache. Ce serveur web a été développé à partir de 1995 en *open source*, et représente aujourd'hui plus de 60 % des serveurs utilisés sur l'internet (Mockus et alii., 2000; Von Hippel, 2002). Le même schéma est applicable à Linux.

commerciales. On a alors une articulation spécifique du marchand et du non-marchand, du monétaire et du non-monétaire, avec un financement indirect des activités, du travail gratuit et du don, autour de la constitution progressive d'une ressource commune. Il n'y a donc pas de concurrence commerciale directe avec les logiciels propriétaires. La dynamique d'un logiciel libre – essor ou déclin - se situe ailleurs, dans le rapport de ce logiciel avec un certain nombre d'utilisateurs-producteurs, et plus généralement avec la demande sociale adressée ou non à une valeur d'usage particulière. Car en matière de logiciel libre, la progression de l'usage accroit le nombre des contributeurs et l'effort de développement, sans médiation marchande ou financière. Les logiciels libres ont alors introduit - ou plus simplement retrouvé - une articulation particulière de la sphère marchande et de la sphère non-marchande de l'économie, une articulation qui reste à l'opposé du modèle commercial propriétaire. Dans celui-ci, les recettes doivent impérativement rembourser ou faire vivre le développement du logiciel. L'échec sur le marché signifiant l'arrêt de tout développement, ce qui ne peut être le cas pour un développement *open source*.

L'articulation marchand/non-marchand se décline cependant de manière différente suivant les situations. Esquissons ici une typologie.

(1) La distribution directe du code-source, rendue obligatoire par les licences. Ce code-source circule parmi les programmeurs-utilisateurs en l'état, et ne peut être utilisé comme exécutable qu'après compilation. Ce code est généralement stocké sur ou plusieurs sites, recensés dans des bases de données que sont *sourceforge*, *freshmeat*, la F.S.F., etc. Il est téléchargeable gratuitement par les utilisateurs, et certains d'entre eux, l'ayant modifié renvoient en sens inverse leurs propres contributions. Cette forme de distribution est en effet faite pour le travail sur le code; et on trouve ici du code "stable" – le logiciel étant suffisamment débogué pour fonctionner correctement – ou plus souvent du code jugé "instable", pour un programme toujours en cours de développment. Dans les grands projets comme Linux, plusieurs versions concurrentes peuvent être ainsi maintenues en parallèle. Depuis 1994 par exemple on distingue les "noyaux stables", où on ne corrige plus que des bogues mineurs, sans toucher aux algorithmes et aux structures de données, et les "noyaux en développement" où l'expérimentation reste beaucoup plus libre. Un système de numérotation permet de s'y retrouver. Le code 2.3.51 donne par exemple le numéro de la version (2) et le numéro de la publication (51); le 3 (nombre impair) indiquant une version en développement. Le code 2.2.14 correspond à une version stable (2 étant pair). Parallèlement, dans d'autres projets logiciels, sont parfois mises en lignes des versions binaires déjà compilées pour les principaux systèmes d'esploitation existants. Cette pratique a pour but de favoriser l'adoption par des utilisateurs finaux (non programmeurs) du programme.

(2) La distribution de paquets logiciels. Entre les projets de développement et les utilisateurs finaux (programmeurs ou non), est apparue une activité intermédiaire d'édition et de distribution de paquets autour du système d'exploitation Linux. On trouve ici des distributions non marchandes, sur le mode du logiciel libre. La plus importante est Debian. Un groupe assez large de développeurs volontaires (plus de 900) a ainsi constitué des ensembles de logiciels libres autour des noyaux Linux (stables), auquel ils ajoutent une interface graphique utilsateur (GUI) comme KDE ou GNOME, un navigateur, une suite bureautique, d'autres logiciels, des jeux, etc., en choisissant ces différents éléments parmi tous ceux qui existent dans le logiciel libre, et en écrivant les liaisons nécessaires à l'ensemble, etc. Les utilisateurs isolés peuvent ainsi disposer de choix tous faits sans avoir à réaliser eux-mêmes tout un travail d'intégration. Ces distributions maintiennent en ligne différents services : archives, forums, mails, suivi et correction des bogues, etc. Les réalisations sont classées suivant le degré de stabilité et le statut légal (pour les USA ou pour le reste du monde, pour des problèmes liés à la cryptographie), téléchargeables directement ou vendues en version CD.

Parallèlement, des distributions marchandes à financement capitaliste classique, employant des salariés, côtées en Bourse, etc., sont apparues ici. Les plus connues sont Redhat, Suse, Mandrake Linux. Elles vendent alors des paquets de logiciels libres (intégrant éventuellement certains logiciels propriétaires), plus les services d'installation, l'aide et la formation, etc. Des offres

différentes à destination des entreprises (petites ou grandes) ou des particuliers, ont conduit à la formation de nombreuses sociétés concurrentes, un peu écrêmées cependant par la crise des *dotcom*.

(3) La formation et l'aide en ligne (ou autrement). Cette activité, liée à l'utilisation des logiciels libres, peut être non-marchande ou marchande, et souvent alors intégrée à d'autres offres. La plus grande partie de ce qui existe ici est en effet le fait de bénévoles, ce qui n'est d'ailleurs pas spécifique aux logiciels libres, la même chose existant dans les univers Windows ou Mac.

(4) L'édition (sur papier) de manuels ou de documentation. Toute cette activité relève d'une activité d'édition commerciale classique. Si les langages et les logicles sont gratuits (et sous GPL), les livres restent inévitablement payants (et généralement sous *copyright* classique), même si parfois une édition en pdf (ou html) peut facilement être disponible sur le web (exemple ici d'O'Reilly).

Le modèle économique du logiciel libre articule donc de manière spécifique le non-marchand et le marchand. Au centre, pour tout ce qui est travail sur le code et circulation de celui-ci, le don et le volontariat domine, et le fiancement de l'ensemble est donc indirect, complètement détaché des activités commerciales. Sur cette ressource, se greffent de multiples activités d'accompagnement (distribution, édition, services, etc.), qui peuvent être gratuites ou payantes, sans possibilité cependant qu'apparaisse réellement ici une position de monopole, et donc le prélèvement d'une rente. Cette économie du logiciel libre est aujourd'hui bien vivante. Loin de se développer à l'écart de l'économie informatique traditionnelle (logiciels compris), elle est plutôt en train de se combiner avec celle-ci dans un double mouvement : d'intégration tout d'abord aux offres et stratégies commerciales des acteurs de l'industrie informatique, mais aussi et plus profondément, d'abandon çà et là du modèle de développement propriétaire en faveur de l'*open source*.

Il existe de nombreuses raisons, techniques, mais aussi économiques (place des logiciels dans l'offre de la firme ou rivalités commerciales) qui poussent à abandonner les stratégies propriétaires pour adopter en tout ou en partie le modèle du logiciel libre[16]. C'est particulièrement le cas quand les logiciels ne sont qu'une partie de l'offre de la firme, l'avantage concurrentiel étant lié à d'autres produits, à la compétence d'une équipe, à la fourniture de services, etc. L'intégration croissante des logiciels libres dans l'offre commerciale des sociétés de services et des fournisseurs de matériel est ainsi une décision parfaitement compréhensible si on prend en compte les avantages du libre pour les utilisateurs.

Il y a d'abord ici aujourd'hui un certain nombre de produits fiables et "bon marché", avec derrière des équipes actives de développeurs, ce qui est de fait une garantie en termes de qualité et de maintenance, sans même parler de la diminution du coût qui peut être parfois fort substantielle. Le code source reste toujours accessible et modifiable. On peut donc le reprendre pour l'améliorer ou en faire autre chose sans avoir à ré-écrire de a jusqu'à z tout un programme, ce qui est appréciable pour certaines opérations de maintenance ou d'ingénierie[17]. Par ailleurs, pour beaucoup de firmes, les logiciels libres ne sont aucunement dangereux en terme de concurrence, ils peuvent au contraire enrichir à peu de frais l'offre, et permettre de contrer le monopole existant (Microsoft par exemple). L'intégration dans l'offre d'IBM destinée aux entreprises du système d'exploitation Linux, d'Apache, de Firefox et d'autres logiciels libres, le soutien (y compris financier) à certains projets de développement *open source*, s'inscrivent alors en partie au moins dans cette logique de la concurrence oligopolistique. C'est une stratégie qui pourrait s'avérer plus efficace que les procédures anti-trust. Par ailleurs, on peut mettre sur pied des offres hybrides en combinant du logiciel libre et du logiciel non libre, à condition de bien séparer les différents composants et les licences respectives. Un exemple ici pourrait être le dernier système d'exploitation de la firme Apple (OS X) avec son noyau Unix sous licence *free software* (projet Darwin), accompagné d'un grand nombre de logiciels GNU-GPL, et son interface graphique propriétaire, qui donne son *look* et son ergonomie à

16 Voir Raymond (1999) sur ce point.

17 Dans le projet de construction d'un super-calculateur pour le CEA (et la force de frappe) par Bull, le système d'exploitation retenu est Linux. Il devrait enrober les différents processeurs Intel fonctionnant en parallèle; ce choix étant justifié par la possibilité d'accèder au code source et d'optimiser ainsi finement la vitesse et les performances du système.

l'ensemble, plus des logiciels (Safari, Mail, TextEdit, etc.) développés par Apple. Cette stratégie, qui permet à Apple d'économiser des coûts de développement, est possible, car les licences la permettent, et qu'il n'y a pas de concurrence commerciale entre Apple et les projets de développement *open source*.

Ces dernières années, on a pu voir alors certains éditeurs de logiciels abandonner sur certains produits la stratégie propriétaire antérieure de protection du code pour mettre celui-ci à la disposition des communautés *open-source*, en optant parfois pour des stratégies semi-propriétaires. L'exemple le plus célèbre est Netscape, mais il en existe d'autres, StarOffice, Blender, etc. La société Netscape, ayant été évincée du marché des navigateurs web par Microsoft (avec Internet Explorer) a ainsi décidé en 1998 de "libérer son code", et permis la mise sur pied d'un projet de développement *open-source*, Mozilla.org, dont le produit phare aujourd'hui est Firefox[18]. Ce projet est accompagné d'une licence spécifique (MPL) et géré par un staff d'ingénieurs salariés, le financement étant assuré par différentes entreprises (Netscape, Redhat, Oeone, Sun, IBM, etc.). Un autre exemple est celui de la suite bureautique Star Office, développée à l'origine par une société allemande et rachetée par Sun en 1999. Sun a poursuivi un temps le développement accompagné d'une diffusion gratuite par téléchargement; puis le code a été "libéré" et mis sous GPL (version 5.2), donnant naissance alors à OpenOffice.org. Deux projets de développement fonctionnent ainsi en parallèle : celui d'OpenOffice sur le mode coopératif et non-marchand de la plupart des logiciels libres, et celui de Sun qui ajoutant des fonctionnalités supplémentaires et des services, revend désormais le tout sous sa marque StarOffice aux entreprises[19].

Il existe donc de nombreuses raisons qui poussent les industriels et les sociétés de services de l'informatique à appuyer aujourd'hui le mouvement des logiciels libres, en intégrant ceux-ci à leur offre, en développant des stratégies hybrides (parfois semi-propriétaires), en libérant le code de certains programmes, en finançant la mise sur pied de projets de développement, en renforçant donc la mise en propriété commune du code. C'est un moyen de couper l'herbe sous le pied aux monopoles existants ou en voie de constitution. L'*open source* répond d'ailleurs directement à un problème récurrent de cette industrie où tout est interconnecté, le besoin de conserver certains standards ouverts, et plus fondamentalement certaines technologies en propriété commune, sous peine de paralysie. Avec le succès du développement de Linux, une partie de l'industrie a aussi commencé à admettre que libérer le code était favorable à l'innovation (une innovation qu'on peut toujours ensuite exploiter commercialement). On voit alors apparaître des discours et des pratiques un peu contradictoires, comme celles d'IBM qui a toujours défendue les brevets logiciels, et dépose d'ailleurs *patent* après *patent*, mais qui soutient aussi les logiciels libres et décide soudain de mettre dans le domaine public 500 brevets logiciels cruciaux pour les communautés de programmeurs.

18 En 1995, Netscape représentait 80 % du marché des navigateurs. L'apparition d'Internet Explorer (Microsoft) intégré dès l'origine à Windows et distribué alors "gratuitement" a conduit à un déclin rapide des ventes et de la part de marché (moins de 12 % du marché en 2001). Libérer le code-source est alors un moyen pour Netscape de prendre sa revanche sur Microsoft en relançant le développement (et l'adoption) d'un produit qui avait perdu tout avenir commercial.

19 OpenOffice (et StarOffice) représentent des alternatives crédibles à Microsoft Office. Pour la société Sun, la libération du code permet de relancer le développement, et de renouer avec des ventes pour un produit qui n'avait plus guère d'avenir commercial (une suite StarOffice est vendue environ 100 $ pour 1000 $ pour Microsoft Office).

Références :

Allen R. C. ,1983, Collective Invention, *Journal of Economic Behavior and Organization*, 4, p. 1-24.

Arrow K., 1962, Economic welfare and the allocation of resources for invention, in *The rate and direction of inventive activity*, Princeton University Press, Princeton , New Jersey.

Browne C. B.*,* 1998, *Linux et le développement décentralisé,* trad. fr. sur http://www.linux-france.org.

Campbell-Kelly, 2003, *Histoire de l'industrie du logiciel, des réservations aériennes à Sonic le hérisson*, MIT, trad. fr. Vuibert.

Clément-Fontaine M., 1999, *Une étude juridique de la Licence Publique Générale GNU*, mémoire de DEA en Droit, http://www.crao.net/gpl/ http://www.crao.net/gpl/

Coris M., 2002, Les sociétés de services en logiciels libres : l'émergence d'un système de production alternatif au sein de l'industrie du logiciel, Université Montesquieu, Bordeaux, IFREDE-E3I, *Document de travail, n° 2002-1*.

DiBona C., Ockman S., Stone M. (eds), 1999, *Open sources – Voices from the open source revolution*, O'Reilly & Associates, Sebastopol, California.

Dréan G., 1996, *L'industrie informatique. Structure, économie, perspectives*, Masson, Paris.

Foray D., Zimmermann J.B., 2001, L'économie du logiciel libre. Organisation coopérative et incitation à l'innovation , *Revue Economique*, 52, numéro hors série, p.77-93.

Ghosh R. A; et alii., 2002, *FLOSS : Free/Libre/Open Source Sofware Study*, MERIT, http://floss.infonomics.nl/report.

Ghosh R. A., David P., 2003, *The nature and composition of the Linux kernel developer community, a dynamic analysis*, http://dxm.org/papers/licks1/licksresults.pdf.

González-Barahona J., Ortuno-Perez M., Quirós H., Centeno González J., Matellán Olivera V., 2002, *Counting potatoes: the size of Debian 2.2*, http://opensource.mit.edu/papers.

Horn F., 2004, *L'Economie des logiciels*, La Découverte, coll. "Repères", Paris.

Jullien N., Zimmermann J.B., 2001, Le logiciel libre : une nouvelle approche de la propriété intellectuelle, GREQUAM, *Document de travail n° 01B06*.

Kim E. E., 2003, An introduction to the open source communities, *Working Paper*, Blue Oxen Associates.

Krishnamurthy S., 2002, *Cave or community ? An empirical examination of 100 mature open source projects*, http://faculty.washington.edu/sandeep.

Mangolte P.-A., 2005, Le "chaudron magique" et "l'invention collective", *Economie Appliquée*, tome LVIII, n° 1, p. 59-83.

Mockus A., Fielding R.T., Herbsleb J., 2000, A case study of open source software development : the Apache server, *Proceedings of ICSE'2000*.

Narduzzo A. et Rossi A., 2003, *Modularity in Action : GNU/Linux and Free/Open Source Software Development Model Unleased*, http://opensource.mit.edu/papers.

Orbiten, 2000, *The Orbiten free software survey*, first edition, http://orbiten.org/ofss/01.html.

Perens B., 1998, La définition de l'open source, in *DiBona et alii*., 1999.

Perline et Noisette T., 2004, *La bataille du logiciel libre, dix clefs pour comprendre*, La Découverte.

Printz J., 1998, *Puissance et limites des systèmes informatisés*, Hermes, Paris

Raymond E., 1999, Le chaudron magique, trad.fr., http://www.linux-france.org.

Rosenberg N., 1976, Technological change in the machine tool industry, 1840-1910, *Perspectives on technology*, Cambridge University Press.

Sagot-Duvauroux J.-L., 2006, *De la gratuité*, Editions de l'Eclat, Paris.

Salus P. H., 1994, *A Quarter Century of UNIX*, Addison-Wesley Publishing Company, Inc.

Smets-Solanes J.-P. et Faucon B.,1999, Logiciels libres. Liberté, égalité, business, Edispher, Paris.

Stallman R., 1998, Le système d'exploitation GNU et le mouvement du logiciel libre, in *DiBona et alii.*, 1999.

Stallman R., 1999, La vraie opposition est entre le logiciel libre et le logiciel propriétaire, *Terminal*, n° 80-81, http://www.terminal.sgdg.org.

Von Hippel E., 2002, Open source projects as horizontal innovation networks – by and for users*, MIT Sloan School of Management Working Paper, n° 4366-02.*

WIDI, 2001, Robles G., Scherder H., Tretkowski I., Weber N., *Who is doing it ? A research on libre software developers*, http://widi.berlios.de/paper/study.html.

Yamagata H., 1997, *Le pragmatiste du logiciel libre : entretien avec Linus Torvalds*, trad. fr. Blondeel, sur http://www.linux-france.org.